Part 5: Grid Data Structures

Do You Know?

Set 10
The source code for the AbstractGrid class is in Appendix D.

1. Where is the isValid method specified? Which classes provide an implementation of this method?
Answer: The isValid method is specified in the Grid interface. From the code, we can find that isValid is a method in class AbsrtactGrid. But the class AbsrtactGrid doesn't implement this method. So this method is specified by interface Grid. And class BoundedGrid and UnBoundedGrid provide this method.

2. Which AbstractGrid methods call the isValid method? Why don't the other methods need to call it?
Answer: The getValidAdjacentLocations method call the isValid method. The other method call getValidAdjacentLocations method to get all the valid location.

3. Which methods of the Grid interface are called in the getNeighbors method? Which classes provide implementations of these methods?
Answer: It call methods get and getOccupiedAdjacentLocations. The get method is implements in BoundedGrid and UnBoundedGrid class. And the getOccupiedAdjacentLocations method is implements in AbstractGrid class.

4. Why must the get method, which returns an object of type E, be used in the getEmptyAdjacentLocations method when this method returns locations, not objects of type E?
Answer: The get method returns the elements of the grid. And the getOccupiedAdjacentLocations method finds the locations that exists critter. The get method can get the element in the location and if there is no element, it returns null. By this, we can judge whether a location is a occupied location.

5. What would be the effect of replacing the constant Location.HALF_RIGHT with Location.RIGHT in the two places where it occurs in the getValidAdjacentLocations method?
Answer: The valid adjacent locations would be those that are north, south, east, and west of the given location. In other words, the number of possible valid adjacent locations would decrease from eight to four.

Set 11

The source code for the BoundedGrid class is in Appendix D.

1. What ensures that a grid has at least one valid location?

Answer: In the constructor of BoundedGrid class, if rows <= 0 or columns <= 0, it will throw an IllegalArgumentException. So the number of rows and columns are at least 1.

2. How is the number of columns in the grid determined by the getNumCols method? What assumption about the grid makes this possible?

Answer:

```
// Note: according to the constructor precondition, numRows() > 0, so
// theGrid[0] is non-null.
return occupantArray[0].length;
```

From problem 1 we know that grid must satisfied rows >= 1. So occupantArray[0] is not null. Which makes occupantArray[0].length possible.

3. What are the requirements for a Location to be valid in a BoundedGrid?

Answer: The row number must greater than or equal to 0 and small than the rows number and the column number must greater than or equal to 0 and small than the columns number.

In the next four questions, let r = number of rows, c = number of columns, and n = number of occupied locations.

4. What type is returned by the getOccupiedLocations method? What is the time complexity (Big-Oh) for this method?

Answer: It returns an ArrayList<Location> type. The time complexity for this method is O(r*c).

5. What type is returned by the get method? What parameter is needed? What is the time complexity (Big-Oh) for this method?

Answer: It returns an E type (Whatever type in that location, and return null if nothing in that location). The time complexity for this method is O(1).

6. What conditions may cause an exception to be thrown by the put method? What is the time complexity (Big-Oh) for this method?

Answer: If the location to put is not valid (out of the grid) or the object which may be put to the location is null, will cause an exception. The time complexity for this method is O(1).

7. What type is returned by the remove method? What happens when an attempt is made to remove an item from an empty location? What is the time complexity (Big-Oh) for this method?

Answer: The remove method returns an E (Whatever type in that location, and return null if nothing in that location). type. The time complexity for this method is O(1).

8. Based on the answers to questions 4, 5, 6, and 7, would you consider this an efficient implementation? Justify your answer.

Answer: Yes. We can see from 4, 5, 6, 7 that the method getOccupiedLocations has most complexity O(r*c) and the other are O(1). So we need to consider the implements of getOccupiedLocations method. We can see from the code that the location which is null is not used. So we don't need to store it. We can use a one-dimensional array to store the used location. And then, we can find the occupied location directly.

Set 12

The source code for the UnboundedGrid class is in Appendix D.

1. Which method must the Location class implement so that an instance of HashMap can be used for the map? What would be required of the Location class if a TreeMap were used instead? Does Location satisfy these requirements?

Answer:

The Location class must implement the hashCode and the equals methods.

If a TreeMap were used, the Comparable interface must be implemented. And the compareTo method should be implemented.

Yes. The location class satisfied all these requirements.

2. Why are the checks for null included in the get, put, and remove methods? Why are no such checks included in the corresponding methods for the BoundedGrid?

Answer: Because for UnboundedGrid class, there are no invalid location. Every location can be use in get, put and remove method which is not null is valid. But these method can't operate a null. A null is not a location.

In BoundedGrid, even if the location is not null, the location may out of grid. Because BoundedGrid has fixed row and column. So in BoundedGrid we should use a isValid method to judge whether the location is not and inside grid.

3. What is the average time complexity (Big-Oh) for the three methods: get, put, and remove? What would it be if a TreeMap were used instead of a HashMap?

Answer:

The average time complexity for the three method: get, put and remove is O(1).

If a TreeMap were used instead of a HashMap, the average time complexity is O(log n). N is the number of occupied location.

4. How would the behavior of this class differ, aside from time complexity, if a TreeMap were used instead of a HashMap?

Answer:

Most of the time, the getOccupiedLocations method will return the occupants in a different order. Keys (locations) in a HashMap are placed in a hash table based on an index that is calculated by using the keys' hashCode and the size of the table. The order in which a key is visited when the keySet is traversed depends on where it is located in the hash table.

A TreeMap stores its keys in a balanced binary search tree and traverses this tree with an inorder traversal. The keys in the keySet (Locations) will be visited in ascending order (row major order) as defined by Location's compareTo method.

5. Could a map implementation be used for a bounded grid? What advantage, if any, would the two-dimensional array implementation that is used by the BoundedGrid class have over a map implementation?

Answer:

Yes. Use a map in a BoundGrid it can get occupied location more effective in getOccupiedLocations method.

When the grid is more occupied location than empty location, the two-dimensional array will cost less store place.

Exercise

1. Answer:

I use a LinkedList<OccupantInCol> with a helper class.

And for class SparseBoundedGrid, the row and column value are used to find the corresponding occupied location. Which is that row value as the index of the list and column value as the key to find corresponding element.

2. Consider using a HashMap or TreeMap to implement the SparseBoundedGrid. How could you use the UnboundedGrid class to accomplish this task? Which methods of UnboundedGrid could be used without change?

Answer: There are many method in UnBoundedGrid class that can directly used in HashMapSparseBoundedGrid class. We can easily find that UnBoundedGrid class is also implements by HashMap. As a result, we only need to add row and column for this class. So the put, get, remove and getOccupiedLocations methods can be used without change.

Fill in the following chart to compare the expected Big-Oh efficiencies for each implementation of the SparseBoundedGrid.

Let r = number of rows, c = number of columns, and n = number of occupied locations

Methods	SparseGridNode version	LinkedList<OccupantInCol> version	HashMap version	TreeMap version
getNeighbors	O(c)	O(c)	O(1)	O(logn)
getEmptyAdjacentLocations	O(c)	O(c)	O(1)	O(logn)
getOccupiedAdjacentLocations	O(c)	O(c)	O(1)	O(logn)
getOccupiedLocations	O(r + n)	O(r + n)	O(n)	O(n)
Get	O(c)	O(c)	O(1)	O(logn)
Put	O(c)	O(c)	O(1)	O(logn)
remove	O(c)	O(c)	O(1)	O(logn)

3. Consider an implementation of an unbounded grid in which all valid locations have non-negative row and column values. The constructor allocates a 16 x 16 array. When a call is made to the put method with a row or column index that is outside the current array bounds, double both array bounds until they are large enough, construct a new square array with those bounds, and place the existing occupants into the new array.

Implement the methods specified by the Grid interface using this data structure. What is the Big-Oh efficiency of the get method? What is the efficiency of the put method when the row and column index values are within the current array bounds? What is the efficiency when the array needs to be resized?

Answer: The Big-Oh efficiency of the get method is O(1).

When the row and column index values are within the current array bounds, the efficiency of the put method is O(1).

When the array needs to be resized, the efficiency of the put method is O(n^2). n is the length before resized.